Automated Analysis of Industrial Workflow-based Models

Mario Cortes-Cornax, Ajay Krishna, Adrian Mos, Gwen Salaün

Automated Analysis of Industrial Workflow-based Models

Mario Cortes-Cornax
Univ. Grenoble Alpes, CNRS, Grenoble INP, LIG, F-38000 Grenoble France

Ajay Krishna
Univ. Grenoble Alpes, Inria, CNRS, Grenoble INP, LIG, F-38000 Grenoble France

Adrian Mos
Naver Labs Europe, Meylan, France

Gwen Salaün
Univ. Grenoble Alpes, CNRS, Grenoble INP, Inria, LIG, F-38000 Grenoble France

ABSTRACT

Modelling and governance of business processes are important concerns in companies all over the world. By better understanding business processes, different optimizations are made possible, concretely resulting into potential efficiency gains, cost reductions and improvements in agility. The use of formal specification languages for the modelling of business processes paves the way for different kinds of automated analysis. Such analysis can be used to infer properties from the modelled processes that can be used to improve their design. In this paper, we particularly explore two important classes of verification, namely verification of behavioural properties using model checking techniques and data-based analysis using SAT solving. Those verifications are fully automated by using different tools such as the CADP verification toolbox and the Z3 solver. We illustrate our approach on a real-world case study.

CCS CONCEPTS

• **Applied computing** → **Business process modeling**; • **Software and its engineering** → **Formal software verification**; • **Theory of computation** → *Logic and verification*;

KEYWORDS

Business process management, SAT, Validation, Data

1 INTRODUCTION

A business process is a collection of structured activities fulfilling a precise goal. Business process management is of prime importance in companies, because they have realized that, by modelling and then mastering their own processes, several improvements could be achieved resulting in time and money savings. Although modelling is necessary to make those optimizations possible, it is unfortunately not enough. Beyond process modelling, there is a need for formal checking that automatically allows one to analyze a process under development or already deployed, and detect whether this process satisfies some precise functional or non-functional requirements. Such verification techniques are helpful to identify possible bottlenecks, missing services or incorrect behaviours. This may then lead to the refinement of the process being analyzed.

In this work, we chose to consider Mangrove [18] for modelling purposes. Mangrove is a meta-model that allows domain specific process languages to be mapped to standard workflow languages, making it easier for non-technical users to design business processes in an intuitive way. Mangrove was used for modelling realistic processes at Xerox[1] as we will show in this paper with a case study.

Given a Mangrove model, we study two kinds of analysis here : the behavioral and the data-based analysis. The first one aims at verifying whether the model satisfies a certain temporal property (*e.g.*, a certain task is never executed after another one). Model checking techniques are used for this analysis. Another kind of behavioural analysis verifies two versions of a process (before and after modification for instance). This is useful to check whether an evolution of a given process respects certain behaviour (a new functionality is present and occurs when expected for example). The aforementioned analysis is achieved using equivalence checking techniques. The data-based analysis looks at the control flow graph that can be derived from the Mangrove model, and check whether all parts of the process are reachable or not. This check is achieved using static analysis of the workflow. Both kinds of verification are automated reusing existing tools (the CADP toolbox [10] and the Z3 solver [2], respectively). Model transformations from Mangrove are developed in order to fill the gap between the industrial process models and the input languages of those tools.

The rest of this paper is organized as follows. Section 2 introduces the Mangrove meta-model. Sections 3 and 4 present the behavioural verification techniques and the data-based analysis, respectively. In Section 5 our approach on an industrial case study is illustrated. Section 6 reviews related work and finally, Section 7 concludes the paper.

2 PROCESS DEFINITION MODELS

Mangrove [18] is a generic process description meta-model that relies on the Eclipse Modelling Framework. It unifies business processes and Service Oriented Architecture (SOA). It provides behavioural support to a domain definition in order to define the necessary steps in a process focusing on preserving the connection between the common elements of business processes and architectural constructs such as services. It facilitates domain specific design by establishing concept mappings between domain related concepts and process activities.

In the approach supported by Mangrove and used in the scenario illustrated in the case study, processes are defined using a combination of inter-related meta-models. The first one, *Domain Meta-model (DomainMM)*, is mainly used to specify domain-specific behaviour (*i.e.*, activity types) and data. There is one instance of this per application domain. For example, our case study deals with

[1]This work was carried out when Mario Cortes-Cornax and Adrian Mos were working at the former Xerox Research Center Europe (XRCE) in Meylan (France), which became Naver Labs.

the Document processing domain, which includes behavioural activities such as scanning, optical character recognition (OCR) and several kinds of quality controls.

The second meta-model, *Common Meta-model (CommonMM)*, is used to define the actual processes and their flow of control, including connections to services. It can be seen as a simplified version of BPMN that connects behaviour (process flows, conditions) and domain definitions (types of behaviour elements, forms and data). A specific implementation of the CommonMM derived from the Mangrove open-source project.

In this paper, we focus on process modelling (supported by the Mangrove CommonMM). The verification approach presented in the next sections deals with the behavioural description of these processes. On the left hand side of Figure 1 some excerpts of processes are given. Notice that the notation relies on steps and conditional sequence flows whereas gateways (*i.e.*, control flow) are implicitly defined. Mangrove models are transformed to the Process Intermediate Format (PIF), which is the input of the analysis tools. PIF and the transformation patterns are described in the next section.

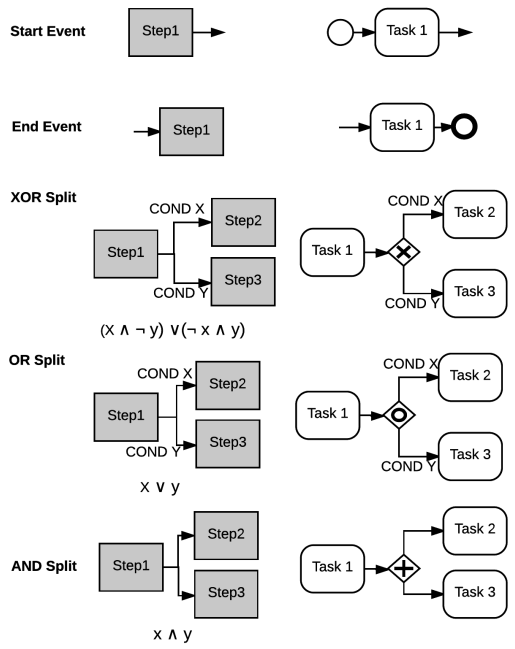

Figure 1: Mangrove to PIF transformation patterns

It is important to note that since the DomainMM and the CommonMM are connected, more complex verifications can leverage this relationship and produce rich domain-specific insights that can help non-technical stakeholders understand the advantages and limitations of various process models in their domain terminology. The connection is realised through model links between a small number of elements. Of particular importance for this are the *DSActivityType* and *DSService* elements in DomainMM. They correspond to domain specific activity types (behaviour) and service definitions. They have individual unique IDs. These IDs are used to make the connection with their flow-oriented elements from the CommonMM: the *Step* that corresponds to the execution of an activity and the *Service* that corresponds to the call of a reusable service. *Steps* are interconnected through *Transition* elements. In addition, the flows are managed at branching points by forking several *TransitionUnderCondition* elements. Data and forms are automatically managed by analysing Mangrove processes and the IDs of their various elements, and by extracting the relevant domain counterparts. Connections between data inputs and outputs are realised based on the types of connected behavioural and service elements. Similarly, forms are displayed based on their occurrence in the domain in the scope of *Step* instances being executed in the Mangrove process.

More information about these meta-models and their intended usage in enterprise settings can be found in [18].

3 BEHAVIOURAL VERIFICATION

In this section, a model transformation from Mangrove to the Process Intermediate Format (PIF) is first presented, and this enables behavioural analysis using the CADP toolbox.

3.1 From Mangrove to PIF

Process Intermediate Format (PIF) [21] is a pivot model for workflow-based notations. A transformation from Mangrove to PIF is proposed in this work, because PIF is supported by the automated analysis techniques that are presented later in this section. The PIF scheme is described in Figure 2, which illustrates its main concepts in a meta-model.

As mentioned before and illustrated in Figure 1, the key elements of Mangrove meta-model that we focus on are *Steps* and *Conditional Sequence Flows*. Mangrove simplifies convergence and divergence of paths by eliminating gateways. Instead of gateways, Mangrove relies on conditions to determine the flow. Condition expressions are evaluated for each flow and tokens are sent across if the condition expression evaluates to *true*. Based on the evaluation of condition expressions, Mangrove flows can be transformed into a PIF gateways. PIF captures the workflow as a set of Nodes and Sequence Flows. Nodes are further classified by their *type* attribute. Gateways and Tasks are the most commonly used node types. Gateways are further classified as Join and Split gateways depending on whether they converge or diverge.

Mangrove to PIF transformation patterns are shown in Figure 1. Each *Step* in Mangrove is translated as a *Task* in PIF. Mangrove models identify outgoing flows from a Step as *Source transitions* and incoming flows as *Target transitions*. The Mangrove to PIF transformations of events and gateways are the following:

Start Event. If there is no target transition for a step, then that Step behaviour is equivalent to a Start event followed by a Task

End Event. If there is no source transition for a step, then that Step is a precursor to End event.

XOR Split Gateway. When a step has multiple source transitions and each transition has a condition associated with it, all the condition expressions are evaluated using SAT techniques. If these expressions evaluate to *true* as an XOR clause ($CondX \oplus CondY$), which means that they are mutually exclusive, then the transformation is identified as an XOR split.

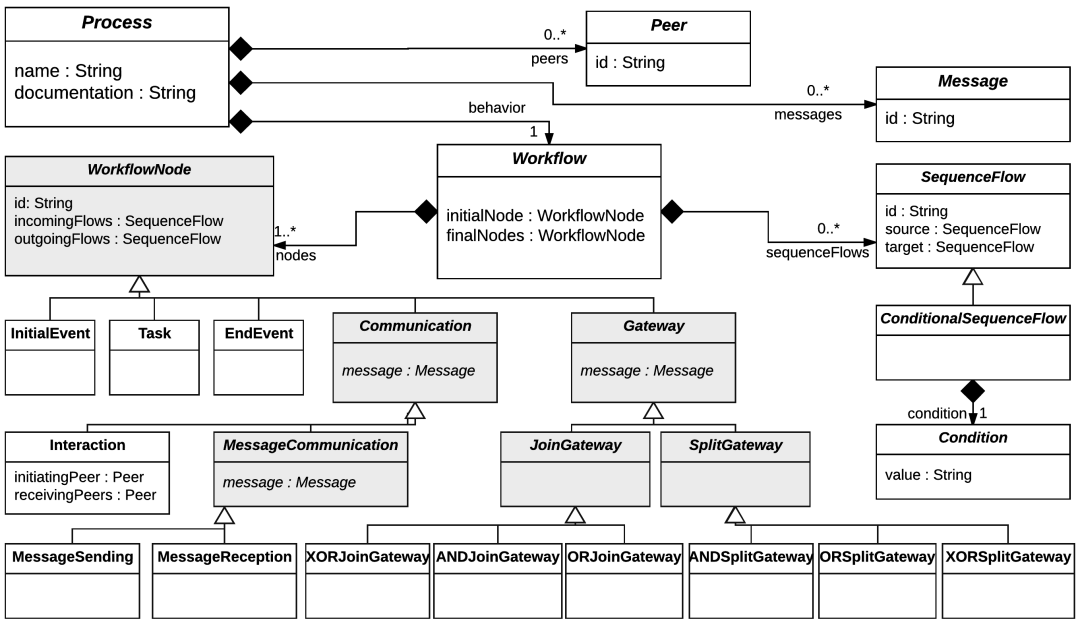

Figure 2: PIF meta-model

OR Split Gateway. This is similar to the XOR pattern as all the condition expressions of source transitions are evaluated. However, in that case, their evaluation shows that the conditions are not exclusive. They evaluate to *true* as an OR clause ($CondX \vee CondY$) transforming to an OR split.

AND Split Gateway. This pattern is pretty straightforward. If there are multiple source transitions for a step and none of them have a condition expression associated with it, then the behaviour is similar to an AND split gateway.

Merge patterns can exist if there are multiple target transitions for a Step. These transitions are transformed to Join gateways in PIF. The type of Join gateway is determined by the preceding Split gateway in the workflow. If the split preceding the join is an Exclusive split (XOR Split), then the ensuing merge pattern is identified as *XOR Join Gateway* by a pile mechanism. Similarly, merge corresponding to an OR Split and an AND Split is treated as *OR Join Gateway* and *AND Join Gateway*, respectively. AND Join Gateway synchronises on all incoming flows before proceeding with outgoing flow.

When a model-to-model transformation is performed, the question of semantics preservation arises. As Mangrove does not propose a formal semantics, this correspondence is difficult to establish. However, since PIF semantics relies on the LNT process algebra [14, 21], our approach allows us to give an implicit semantics to Mangrove by translation to PIF.

3.2 Behavioural Analysis

PIF is supported by the VBPMN platform [14, 21], which connects PIF to the CADP toolbox. This is achieved by translating a PIF process into LOTOS New Technology (LNT) code that is one of the input specification languages of the CADP toolbox [10]. The CADP toolbox can be reused for generating an LTS (Labelled Transition System). The latter describes all the executions of the workflow starting from the initial state, using a set of states, transitions and labels. Labels correspond to the steps in the Mangrove model. CADP is also used for verifying the two following classes of properties on LTS models obtained from Mangrove proces models:

- *Functional verification* aims at checking properties of interest such as the existence of deadlock/livelock states or the satisfaction of safety and liveness properties. In the latter case, since the properties depend on the input process, they have to be provided by the analyst, who can reuse well-known patterns for properties such as those presented in [8].
- *Process comparison* takes as input two process models, a comparison relation and possibly additional parameters for the relation. Several evolution relations can be used. Conservative evolution ensures that the observational behaviour is strictly preserved. Inclusive evolution ensures that a subset of a process behaviour is preserved in a new version of it. Selective evolution allows one to focus on a subset of the process tasks. It is also possible to have VBPMN work up-to a renaming relation over tasks. If the two input process models do not fulfil the constraints of the chosen evolution relation, a counterexample indicating the source of the violation is returned. This helps the process analyst in supporting the refinement into a correct evolved version of a process model.

4 DATA-BASED ANALYSIS USING SAT

In the former section, the use of model and equivalence checking techniques are proposed to verify workflow behaviour. The LTS generated using these techniques is an over approximation consisting of all possible executions and it is not the exact representation

of the workflow behaviour. This is caused by the fact that the verification does not take into account data information encoded in the workflows, which would make certain possible execution paths invalid. Workflow models like Mangrove and BPMN have conditions encoded in sequence flows. If these conditions are evaluated, we can easily identify if the corresponding execution path is feasible. As these models can be viewed as a control flow graph, a static analysis of the model can be performed by encoding condition expressions as satisfiability constraints. Satisfiability (SAT) and satisfiability modulo theories (SMT) [3, 4] can be applied to check if the execution paths are satisfiable. Satisfiability theory is based on solving propositional formulas. It works on the premise that a formula is satisfiable, if there exists a set of values that evaluate to *true*. We opted to build satisfiability constraints, rather than evaluating condition expressions step by step programmatically, as it allows us to leverage the power of high-performance SMT solvers like Z3 [2] to solve large sets of condition expressions. Note that we have already used these techniques in Section 2 in order to check whether a split pattern transforms to an XOR or OR split gateway.

Figure 3 shows a Mangrove workflow with two instance variables *age* and *expense* (*exp* for short) and their condition expressions. Using model checking techniques, we can generate all possible executions without considering those conditions. Suppose the dataset under consideration has expense values in the range $124 < exp < 679$, we can refine the earlier over-approximate model, by discarding the bottom half of the workflow. This optimisation might not seem significant for small workflows, but for large workflows, especially with many OR gateways, when behaviour becomes too complex, refining it helps to understand the model better. Another benefit from refining is that we can eliminate "false negatives" in verification (*i.e.,* we might detect anomalies in parts of the workflow, which would not be executed for the given dataset). Respecting the execution semantics, data in Figure 3 can be expressed as a propositional formula as follows:

$$((exp \geq 100) \wedge (age > 30 \vee age = 35 \vee age < 40)) \\ \vee \neg(exp \geq 100)$$

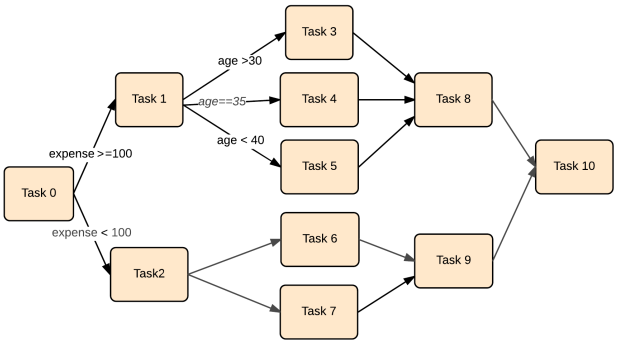

Figure 3: Workflow with data

In our approach, the input dataset is added as an initial constraint. Starting from the first *Step* in the workflow, constraints are incrementally added and removed as condition expressions are encountered. As the formula is updated, its satisfiability is checked - if the result is satisfiable, we proceed further, otherwise, the state is marked as infeasible and we backtrack to see if any other diverging path is satisfiable. Once we have identified infeasible flows, we generate an updated PIF model, which can serve as new input for behavioural verification (Section 3).

The constraint building process can be illustrated using an input dataset. For the input dataset $exp = [125, 678]$ and $age = [45, 90]$, after the execution of *Start Event*, the following propositional formula would be generated:

$$((exp \geq 125 \wedge exp \leq 678) \wedge (age \geq 45 \wedge age \leq 90)) \quad \text{(i)}$$

Further, as we traverse the *Exclusive gateway* conditions, we obtain another constraint, in addition to formula (i). Thus the two formulas are as follows:

$$((exp \geq 125 \wedge exp \leq 678) \wedge (age \geq 45 \wedge age \leq 90) \\ \wedge(exp \geq 100)) \quad \text{(ii)}$$

$$((exp \geq 125 \wedge exp \leq 678) \wedge (age \geq 45 \wedge age \leq 90) \\ \wedge(exp < 100)) \quad \text{(iii)}$$

Clearly, formula (iii) is not satisfiable, thus we can identify *Task 2* as infeasible for the input data set. So, we can build on formula (ii), which is satisfiable, and following propositional formulas can be derived:

$$((exp \geq 125 \wedge exp \leq 678) \wedge (age \geq 45 \wedge age \leq 90) \\ \wedge(exp \geq 100) \wedge (age > 30)) \quad \text{(iv)}$$

$$((exp \geq 125 \wedge exp \leq 678) \wedge (age \geq 45 \wedge age \leq 90) \\ \wedge(exp \geq 100) \wedge (age = 35)) \quad \text{(v)}$$

$$((exp \geq 125 \wedge exp \leq 678) \wedge (age \geq 45 \wedge age \leq 90) \\ \wedge(exp \geq 100) \wedge (age < 40)) \quad \text{(vi)}$$

Similarly, if we solve (iv), (v) and (vi) for satisfiability, only (iv), is satisfiable, so we can mark *Task 4* and *Task 5* as infeasible. The proposed incremental approach can be extended to large workflows.

Here, we have illustrated our approach using an input dataset. We can also check the feasibility of execution of *Steps* in Mangrove by taking only the conditions as constraints (infinite domains). By adding the input dataset as a constraint, we are just restricting the domain of the instance variables to a finite value.

5 CASE STUDY

In this section, a real-world Mangrove model is described and we illustrate how automated analysis can be performed using our approach. The context of this (simplified) use-case is the document treatment process aiming at scanning and extracting information from physical documents submitted by clients using optical character recognition (OCR) and sent it back to clients after treatment. The OCR process is illustrated in Figure 4. When the physical document is scanned a numerical image of the document is created. The OCR transforms the image into actual text, numbers or images. It

is also able to identify the main sections of the document. Finally, the keyword extraction identifies important information based on predefined rules (*e.g.*, the most repeated word, all the numbers followed by a euro symbol or names).

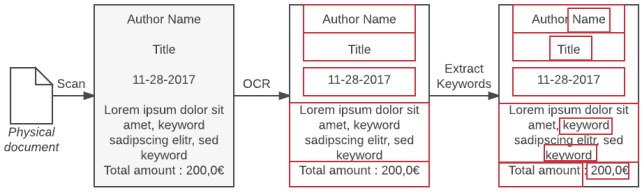

Figure 4: Optical character recognition

Now let us focus on the whole process introduced in Figure 5. When a document is received, it is scanned and then goes through an OCR analysis. After the OCR, a control is performed in order to verify the treatment quality relying on a predefined quality indicator (qInd). This quality indicator is defined as a process variable, which is set by the OCR's output. It aggregates information about the brightness of the document, the number of extracted sections or the confidence in the extracted amounts, and scales it from 0 to 100. Three possible thresholds led to exclusive paths: 1) if *qInd =< 75* then the OCR will be rejected. The rejection is logged and the process loops (maximum twice) to scan again; 2) if *qInd >= 96* then the quality is considered enough to execute a parameterized keyword extraction and finally transfer the scanned file as well as the extracted information back to the client; and 3) if *qInd > 75 and qInd < 96* then a video coding process is performed consisting in a manual verification and validation of the extracted information. A worker verifies on a screen whether the extracted information actually corresponds to the text or numbers of the scanned document. After these manual steps, the keyword extraction and transfer can be performed. Note that the expected quality of service is negotiated with the client through service level agreements.

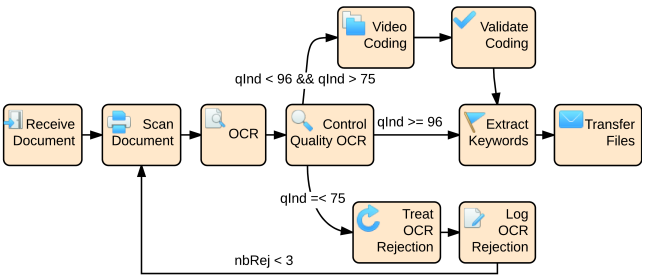

Figure 5: Process 1 - Document processing use-case

Figure 6 proposes an evolution of the process in Figure. 5. The first process has to be improved introducing a rejection treatment just after the reception. Indeed, some documents can be automatically rejected before performing the whole process (*e.g.*, non registered type of documents). The treatment and consistent notification avoids unuseful effort, which could just be detected once the process has completed. Note that there are three levels of rejection after an automatic process: 1) if *rej = 0* then the document is not rejected; 2) if *rej= 1* then there is an immediate rejection; and 3) if *rej = 2* then a manual verification is needed, which led to the normal path (*Scan Document*) or a rejection. We will show on these two versions of the Mangrove process how evolution and property verification as well as data-based analysis are useful.

Behavioural verification. Process 1 (Figure 5) and 2 (Figure 6) can be compiled using CADP into LTS models as shown in Figures 7 and 8. Given a process, the corresponding LTS exhibits all possible executions of that process. The generated LTS takes into account the conditions involved. Since the multiple source transitions have mutually exclusive conditions, the generated LTS has diverging paths. These three exclusive paths are present in the LTS (Figure 8) after *TreatRejection*. When checking for evolution and comparing these models, we first note that the extended workflow does not strictly preserve the previous execution scenarios. This behavioural analysis returns a counterexample, as shown in Figure 9, which indicates a path highlighting the difference between the two models. This information is useful, particularly when the models are large with plenty of possible execution scenarios. As far as automated analysis is concerned, various modes of evolution checking can be used depending on the user needs [21].

The aforementioned LTSs are also used to automatically verify the reachability from one step to another step in the process. As the process shows several possible paths and therefore executions, a designer may need an extra help in order to validate a proposed evolution. For instance, an important verification aims at checking that all the rejections produced in the second workflow are notified to the client. It can be checked using property based functional verification. Figure 10 shows a path returned by the model checker where this property is violated. By taking a closer look at the path, the analyst can see that after rejection, the path ends up by a successful transfer of files. Further, the analyst can either decide that a rejection notification was not necessary in that case, or can update the workflow to avoid an execution where a rejection is decided whereas the behaviour terminates succesfully.

Data-based analysis. The verification has focused so far on the workflow behaviour. In addition, SAT based analysis can be performed to look more carefully at data (conditions) and further optimize the execution scenarios. In the process given in Figure 6, it is possible that a particular input dataset is of registered type ($rej = 0$) and of high quality ($qInd \geq 96$). For this dataset, SAT based analysis would simplify the workflow behaviour to a much simpler LTS as shown in Figure 11. SAT analysis would result in the following constraints being unsatisfiable:

$$((rej = 0 \oplus rej = 1 \oplus rej = 2) \wedge (qInd \leq 75))$$

$$((rej = 0 \oplus rej = 1 \oplus rej = 2) \wedge (qInd > 75 \wedge qInd < 96))$$

It is worth noting that without executing the process workflow, through static analysis, we can identify infeasible paths. This information can be valuable for resource scheduling. In the second version of our workflow, given in Figure 6, the top part (video coding) and the bottom part (OCR rejection treatment) are not executed, and the corresponding resources can thus be freed.

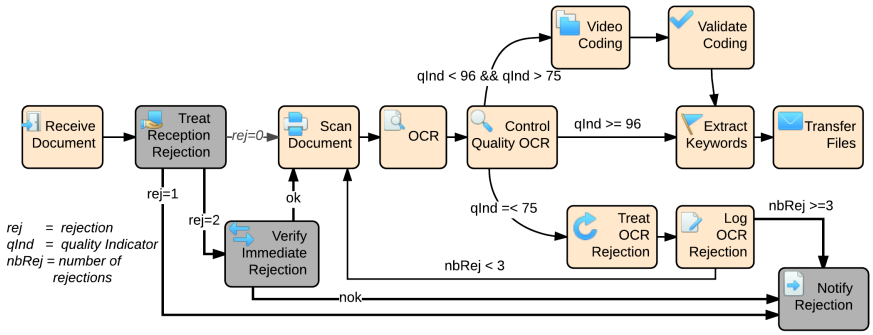

Figure 6: Process 2 - Use case with rejection treatment evolution

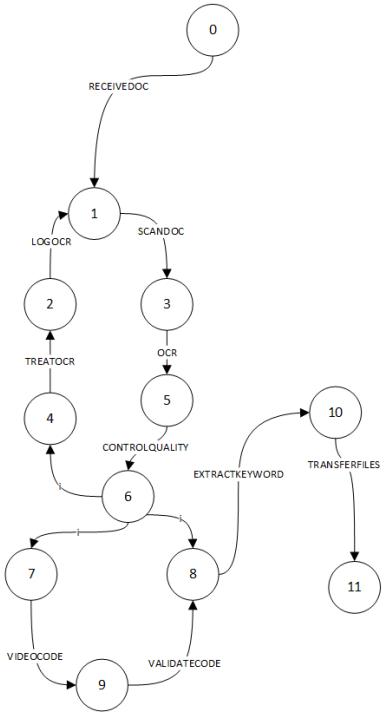

Figure 7: LTS of process 1

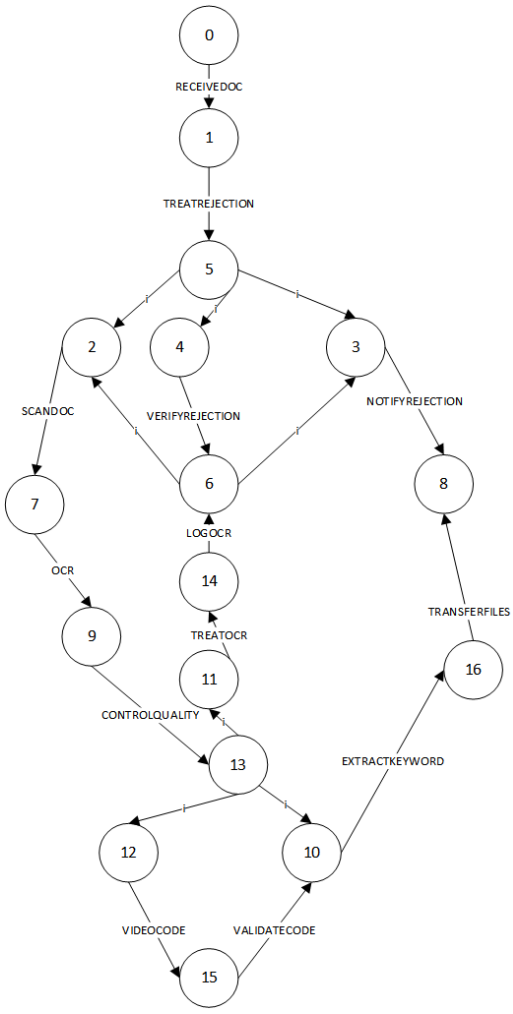

Figure 8: LTS of process 2

6 RELATED WORK

Several works have focused on providing formal semantics and verification techniques for business processes using Petri nets, process algebras, or abstract state machines, see, *e.g.*, [5–7, 11–13, 16, 17, 20, 22, 25, 26]. The main difference in this paper is that we do not only support automated analysis of specific properties but also other kinds of verification (comparison, data-based analysis).

As far as process comparison is concerned, in Chapter 9 of [23], the authors study the migration of processes, and from that point of view define several notions of evolution, migration, and refactoring. We propose off-line analysis techniques and do not propose a solution for applying these changes at runtime. In [24], the authors address the equivalence or alignment of two process models. To do so, they check whether correspondences exist between a set of activities in one model and a set of activities in the other model. They consider Petri net systems as input and process graphs as low-level formalism for analysis purposes. Their approach relies on the identification of regions (set of activities) in each graph that can

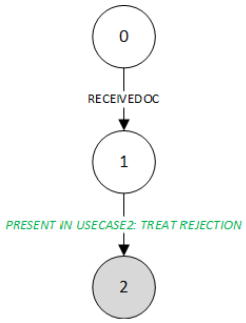

Figure 9: Evolution counterexample

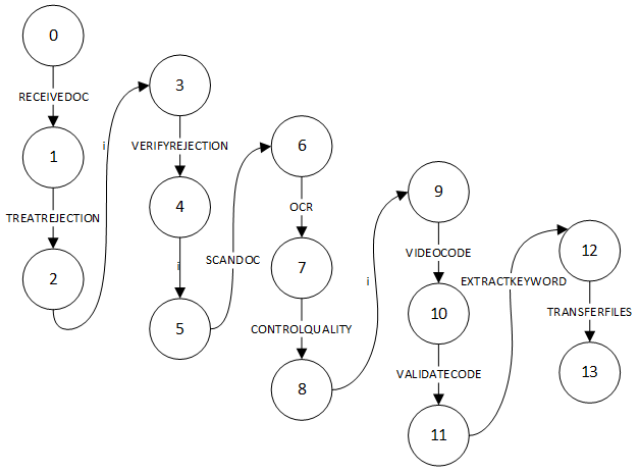

Figure 10: Property verification: Path without rejection notification

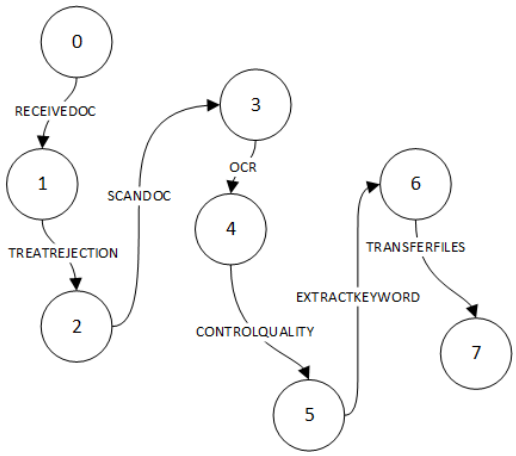

Figure 11: Reachable path LTS

match with respect to an equivalence notion. ADDiff [15] proposes a semantic differencing operator for activity diagrams. As a result, ADDiff performs a semantic comparison and outputs a set of diff witnesses, each of which is an execution trace that is possible in the first activity diagram and is not possible in the second. This solution uses search and fixpoint algorithms whereas we rely on equivalence checking and concurrency theory principles.

As far as data-based analysis is concerned, in [9], the authors propose a translation of BPMN into logic with a special focus on data objects and data-based decision gateways. They provide new mechanisms to avoid structural issues in workflows such as flow divergence by introducing the notion of well-formed BPMN process. Their approach aims at avoiding incorrect syntactic patterns whereas we propose automated analysis at the semantic level.

[19] focuses on the analysis of choreography models. The main property of interest in that context is called *conformance* and aims at checking whether the distributed implementation and the choreography behave identically. The authors mainly focus on data description. Their approach supports choreographies extended with conditions and relies on SMT solving for conformance checking.

Decision Model and Notation (DMN) is a recent OMG standard for modelling decisions in an interchangeable format. DMN can be used into workflow-based notations for representing conditions. In [1], the authors propose a formal semantics of DMN decision tables, a notion of DMN table correctness, and algorithms that check the detection of overlapping rules and missing rules. These algorithms have been implemented in the DMN toolkit and validated through empirical evaluation. Our modelling language for describing decisions is different than DMN since we have to handle infinite domains, justifying our choice of SMT solving.

7 CONCLUSION

This paper studies the crucial issue of business process modelling and analysis. As modelling language, we have opted for a workflow-oriented domain-specific language (Mangrove), which is simple and expressive enough for dealing with the basic constructs of processes without considering implementation details. Regarding verification, we have focused on that question from a functional point of view. Two kinds of verification techniques are proposed for Mangrove, namely *behavioural analysis* and *data-based verification*. They are helpful in analysing the behaviour of a process model in order to identify possible errors such as violated properties or unreachable paths. The detection of such erroneous behaviours may lead to possible improvements in the subsequent versions of the process. In order to automate these checks, we rely on model transformation and the reuse of existing verification frameworks, implementing model/equivalence checking techniques and static analysis. Focusing on an industrial case study, we illustrate how our approach can be used in practice for comparing formally two versions of a process. Our main perspective for future work is to deal with non-functional properties such as execution time, cost analysis or optimal resource allocation.

REFERENCES

[1] D. Calvanese, M. Dumas, U. Laurson, F. Maria Maggi, M. Montali, and I. Teinemaa. 2016. Semantics and Analysis of DMN Decision Tables. In *Proc. of BPM'16 (LNCS)*, Vol. 9850. Springer, 217–233.

[2] L. De Moura and N. Bjørner. 2008. Z3: An Efficient SMT Solver. In *Proc. of TACAS'08 (LNCS)*, Vol. 4963. Springer, 337–340.

[3] L. De Moura and N. Bjørner. 2011. Satisfiability Modulo Theories: Introduction and Applications. *Commun. ACM* 54, 9 (2011), 69–77.

[4] L. de Moura, B. Dutertre, and N. Shankar. 2007. A Tutorial on Satisfiability Modulo Theories. In *Proc. of CAV'07 (LNCS)*, Vol. 4590. Springer, 20–36.

[5] G. Decker and M. Weske. 2011. Interaction-centric Modeling of Process Choreographies. *Information Systems* 36, 2 (2011), 292–312.

[6] R.M. Dijkman, M. Dumas, and C. Ouyang. 2008. Semantics and Analysis of Business Process Models in BPMN. *Inf. Softw. Technol.* 50, 12 (2008), 1281–1294.

[7] F. Durán and G. Salaün. 2017. Verifying Timed BPMN Processes Using Maude. In *Proc. of COORDINATION'17 (LNCS)*, Vol. 10319. Springer, 219–236.

[8] M. B. Dwyer, G. S. Avrunin, and J. C. Corbett. 1999. Patterns in Property Specifications for Finite-State Verification. In *Proc. of ICSE'99*. ACM, 411–420.

[9] N. El-Saber and A. Boronat. 2014. BPMN Formalization and Verification using Maude. In *Proc. of BM-FA'14*. ACM, 1–8.

[10] H. Garavel, F. Lang, R. Mateescu, and W. Serwe. 2013. CADP 2011: A Toolbox for the Construction and Analysis of Distributed Processes. *STTT* 2, 15 (2013), 89–107.

[11] M. Güdemann, P. Poizat, G. Salaün, and A. Dumont. 2013. VerChor: A Framework for Verifying Choreographies. In *Proc. of FASE'13 (LNCS)*, Vol. 7793. Springer, 226–230.

[12] M. Güdemann, P. Poizat, G. Salaün, and L Ye. 2016. VerChor: A Framework for the Design and Verification of Choreographies. *IEEE Trans. Services Computing* 9, 4 (2016), 647–660.

[13] F. Kossak, C. Illibauer, V. Geist, J. Kubovy, C. Natschläger, T. Ziebermayr, T. Kopetzky, B. Freudenthaler, and K.-D. Schewe. 2014. *A Rigorous Semantics for BPMN 2.0 Process Diagrams.* Springer.

[14] A. Krishna, P. Poizat, and G. Salaün. 2017. VBPMN: Automated Verification of BPMN Processes. In *Proc. of IFM'17 (LNCS)*, Vol. 10510. Springer, 323–331.

[15] S. Maoz, J. O. Ringert, and B. Rumpe. 2011. ADDiff: Semantic Differencing for Activity Diagrams. In *Proc. of SIGSOFT/FSE'11*. ACM, 179–189.

[16] A. Martens. 2005. Analyzing Web Service Based Business Processes. In *Proc. of FASE'05 (LNCS)*, Vol. 3442. Springer, 19–33.

[17] R. Mateescu, G. Salaün, and L. Ye. 2014. Quantifying the Parallelism in BPMN Processes using Model Checking. In *Proc. of CBSE'14*. ACM, 159–168.

[18] A. Mos and M. Cortes-Cornax. 2016. Business Matter Experts do Matter: A Model-Driven Approach for Domain Specific Process Design and Monitoring. In *Proc. of BPM Forum'16 (LNBIP)*, Vol. 260. Springer, 210–226.

[19] H. N. Nguyen, P. Poizat, and F. Zaïdi. 2012. A Symbolic Framework for the Conformance Checking of Value-Passing Choreographies. In *Proc. of ICSOC'12 (LNCS)*, Vol. 7636. Springer, 525–532.

[20] P. Poizat and G. Salaün. 2012. Checking the Realizability of BPMN 2.0 Choreographies. In *Proc. of SAC'12*. ACM, 1927–1934.

[21] P. Poizat, G. Salaün, and A. Krishna. 2016. Checking Business Process Evolution. In *Proc. of FACS'16 (LNCS)*, Vol. 10231. Springer, 36–53.

[22] I. Raedts, M. Petkovic, Y. S. Usenko, J. M. van der Werf, J. F. Groote, and L. Somers. 2007. Transformation of BPMN Models for Behaviour Analysis. In *Proc. of MSVVEIS'07*. 126–137.

[23] M. Reichert and B. Weber. 2012. *Enabling Flexibility in Process-Aware Information Systems - Challenges, Methods, Technologies.* Springer.

[24] M. Weidlich, R. M. Dijkman, and M. Weske. 2012. Behaviour Equivalence and Compatibility of Business Process Models with Complex Correspondences. *Comput. J.* 55, 11 (2012), 1398–1418.

[25] P.Y.H. Wong and J. Gibbons. 2008. A Process Semantics for BPMN. In *Proc. of ICFEM'08 (LNCS)*, Vol. 5256. Springer, 355–374.

[26] P.Y.H. Wong and J. Gibbons. 2008. Verifying Business Process Compatibility. In *Proc. of QSIC'08*. IEEE, 126–131.